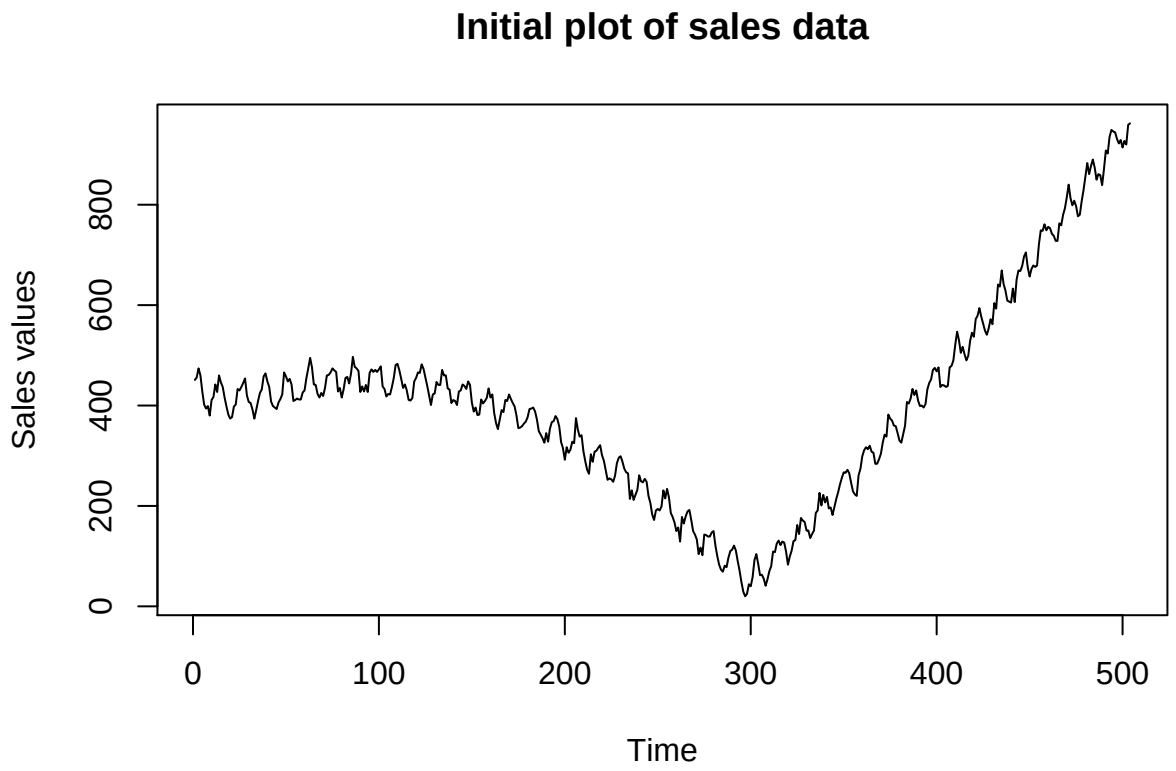
Statistical Methods Coursework Draft

Problem: Analysis of Product Sales data set

We begin by plotting our data.

```
##Set working directory
setwd("~/Documents/SurreyOfUniversity/Y3S2/Statistical_Methods/Assessed_Coursework")

##Add packages
##install.packages("matlib")
##Import data
totaly_real_data <- read.table("Assessed Data 20-21.txt")
sales = totaly_real_data$V1
##Create a dataframe to hold objects
df <- data.frame(totaly_real_data)
colnames(df) <- c("Sales")
```

```
#Plot the data
ts.plot(df$Sales, xlab = 'Time', ylab = 'Sales values')
title("Initial plot of sales data")
```




```
##Due to the size of the graph I won't use the "points()" function
```

We will consider a model of the form $X_t = m_t + s_t + Z_t$ where m_t is the trend s_t is the seasonality and Z_t is some white noise proccess

QUESTION 1 a)

We first consider the trend as a global quadratic polynomial of the form $\hat{m}_t = \hat{\beta}_0 + \hat{\beta}_1 t + \hat{\beta}_2 t^2$. To messure how "close" our guess is to the true answer we consider the cost function

$$S = \sum_{i=1}^{n} (x_t - (\hat{\beta}_0 + \hat{\beta}_1 t + \hat{\beta}_2 t^2))^2$$

(Dont confuse S the cost function and s_t the seasonaity at t). To find our parameters $\{\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2\}$ we must solve the normal equations

$$\frac{dS}{d\hat{\beta}_i} \text{ for } i = \{0, 1, 2\}$$

Thses are given by

$$\frac{dS}{d\hat{\beta}_0} = -2(\sum_{i=1}^{n} x_t - \sum_{i=1}^{n} \hat{\beta}_0 - \sum_{i=1}^{n} \hat{\beta}_1 t - \sum_{i=1}^{n} \hat{\beta}_2 t^2) = 0 \implies \sum_{i=1}^{n} x_t - \sum_{i=1}^{n} \hat{\beta}_0 - \sum_{i=1}^{n} \hat{\beta}_1 t - \sum_{i=1}^{n} \hat{\beta}_2 t^2 = 0$$

$$\frac{dS}{d\hat{\beta}_1} = -2(\sum_{i=1}^{n} t x_t - \sum_{i=1}^{n} \hat{\beta}_0 t - \sum_{i=1}^{n} \hat{\beta}_1 t^2 - \sum_{i=1}^{n} \hat{\beta}_2 t^3) = 0 \implies \sum_{i=1}^{n} t x_t - \sum_{i=1}^{n} \hat{\beta}_0 t - \sum_{i=1}^{n} \hat{\beta}_1 t^2 - \sum_{i=1}^{n} \hat{\beta}_2 t^3 = 0$$

$$\frac{dS}{d\hat{\beta}_2} = -2(\sum_{i=1}^{n} t^2 x_t - \sum_{i=1}^{n} \hat{\beta}_0 t^2 - \sum_{i=1}^{n} \hat{\beta}_1 t^3 - \sum_{i=1}^{n} \hat{\beta}_2 t^4) = 0 \implies \sum_{i=1}^{n} t^2 x_t - \sum_{i=1}^{n} \hat{\beta}_0 t^2 - \sum_{i=1}^{n} \hat{\beta}_1 t^3 - \sum_{i=1}^{n} \hat{\beta}_2 t^4 = 0$$

We can re-write this linear system in matrix form and ask R to solve it for as. The system is given by

$$\begin{bmatrix} \sum_{i=1}^{n} 1 & \sum_{i=1}^{n} t & \sum_{i=1}^{n} t^2 \\ \sum_{i=1}^{n} t & \sum_{i=1}^{n} t^2 & \sum_{i=1}^{n} t^3 \\ \sum_{i=1}^{n} t^2 & \sum_{i=1}^{n} t^3 & \sum_{i=1}^{n} t^4 \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \hat{\beta}_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} x_t \\ \sum_{i=1}^{n} x_t t \\ \sum_{i=1}^{n} x_t t^2 \end{bmatrix}$$

QUESTION 1 b)

We now solve for the $\hat{\beta}s$ first by solving the linear system and then using the linear fit function, but before we solve this system we must find all of the values we need.

```
##We find the sums of the powers of t between 1 and 4.
##We create a vector containing each power, which we shall iterate over.
powers = c(0,1,2,3,4)
##We initialise an empty vector to store the sums
##and loop over the powers and sum over all sales.
sums_of_powers_of_t = c(0,0,0,0,0)
for (i in c(1,2,3,4,5)){
  for (t in 1:length(df$Sales)){
    sums_of_powers_of_t[i] = sums_of_powers_of_t[i]+t^(powers[i])
  }
}

##We also find the sums over (t^n * x_t) for n = (0,1,2)
##As before
n_powers = c(0,1,2)
sums_of_powers_of_t_x_t = c(0,0,0)
for (i in c(1,2,3)){
  for (t in 1:length(df$Sales)){
    sums_of_powers_of_t_x_t[i] = sums_of_powers_of_t_x_t[i]+(t^(powers[i]))*df$Sales[t]
```

```
  }
}
b = sums_of_powers_of_t_x_t
```

We create the matrix of powers of t.

```
##We create the matrix A of the powers of t
##which we will use to solve for the betas
A = matrix(c(sums_of_powers_of_t[1],sums_of_powers_of_t[2],sums_of_powers_of_t[3],
             sums_of_powers_of_t[2],sums_of_powers_of_t[3],sums_of_powers_of_t[4],
             sums_of_powers_of_t[3],sums_of_powers_of_t[4],sums_of_powers_of_t[5]),3,3)
A
```

```
##          [,1]        [,2]         [,3]
## [1,]      504      127260 4.280178e+07
## [2,]   127260    42801780 1.619511e+10
## [3,] 42801780 16195107600 6.536337e+12
```

Next we use the R function "solve" to find the βs, which are given below.

```
##A is the matrix of powers of t
##sums_of_powers_of_t_x_t is our "b" vector
betas <- solve(A, b)
betas
```

```
## [1] 648.997859693  -3.793496346   0.008355363
```

The so we find that $\{\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2\} = \{648.997859693, -3.793496346, 0.008355363\}$. Using R's built in linear fitting function we verify our answer and continue.

```
##We create an index vector for a global quadratic fit
T = 1:length(df$Sales)

##Use the lm function to verify our own values beta values
quadfit <- lm(df$Sales ~ T + I(T^2))
quadfit
```

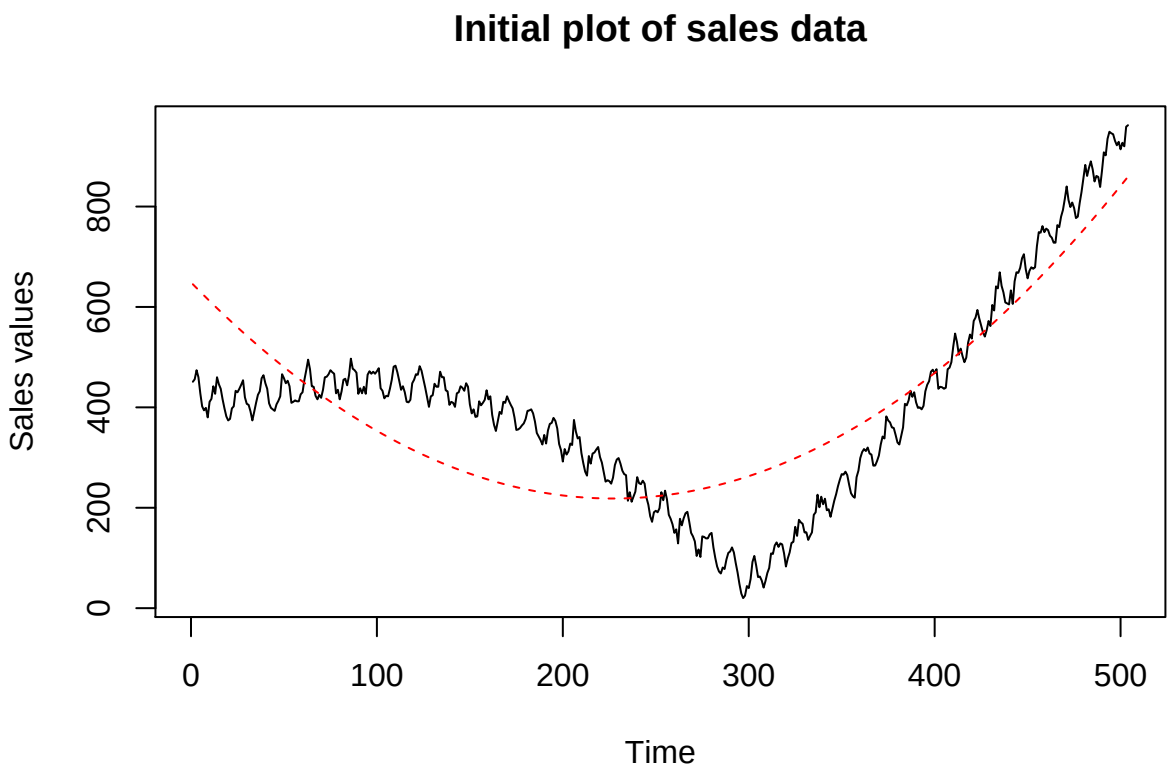
```
##
## Call:
## lm(formula = df$Sales ~ T + I(T^2))
##
## Coefficients:
## (Intercept)            T       I(T^2)
##  648.997860    -3.793496     0.008355
```

```
##Add quadfit to the dataframe
df[,2] = quadfit$fitted.values
##Set new column names
colnames(df) <- c("Sales","QuadFitValues")
```

Great, the values we have obtain match those given by the linear fit function. We now plot the data to see how the fit looks.

```
##Plot the data
ts.plot(df$Sales,
        xlab = 'Time',
        ylab = 'Sales values')
title("Initial plot of sales data")
lines(df$QuadFitValues,
```

```
      col = "red",
      lty = 2)
```



This model matches the data for very few observations and thus is not a good fit. To improve this we might attempt to fit a higher order polynomial or manipulate the data before fitting the polynomial, for example taking its log, but as the powers grow so does the chance of over-fitting the data.

QUESTION 2 a)

Next we consider estimating the trend using a local linear polynomial on successive groups of 5 observations.

For notational simplicity we write the trend at the $(t+j)th$ position by $\hat{m}_{t+j} = \hat{\beta}_0 + \hat{\beta}_1 j, \ \forall\, j \in \{-2, -1, 0, 1, 2\}$

Again we consider the least squares cost function

$$S = \sum_{j=-2}^{2} (x_{t+j} - \hat{\beta}_0 + \hat{\beta}_1 j)^2$$

and obtain the normal equations

$$\frac{dS}{d\hat{\beta}_0} = -2(\sum_{j=-2}^{2} x_{t+j} - \sum_{j=-2}^{2} \hat{\beta}_0 - \sum_{j=-2}^{2} \hat{\beta}_1 j) = 0$$

and

$$\frac{dS}{d\hat{\beta}_1} = -2(\sum_{j=-2}^{2} x_{t+j} j - \sum_{j=-2}^{2} \hat{\beta}_0 j - \sum_{j=-2}^{2} \hat{\beta}_1 j^2) = 0$$

We notice that the sums of the odd power of j goes to 0 in both equations and we then find

$$\sum_{j=-2}^{2} x_{t+j} - \hat{\beta}_0 \sum_{j=-2}^{2} 1 = 0$$

and

$$\sum_{j=-2}^{2} x_{t+j} j - \hat{\beta}_0 \sum_{j=-2}^{2} j^2 = 0$$

by calculation we have $\sum_{j=-2}^{2} 1 = 5$ and $\sum_{j=-2}^{2} j^2 = 10$, rearranging for $\hat{\beta}_0$ and $\hat{\beta}_1$ give us

$$\sum_{j=-2}^{2} x_{t+j} = 5\hat{\beta}_0$$

and

$$\sum_{j=-2}^{2} x_{t+j} j = 10\hat{\beta}_1$$

We know this is the best local linear polynomial because it minimizes the cost function.

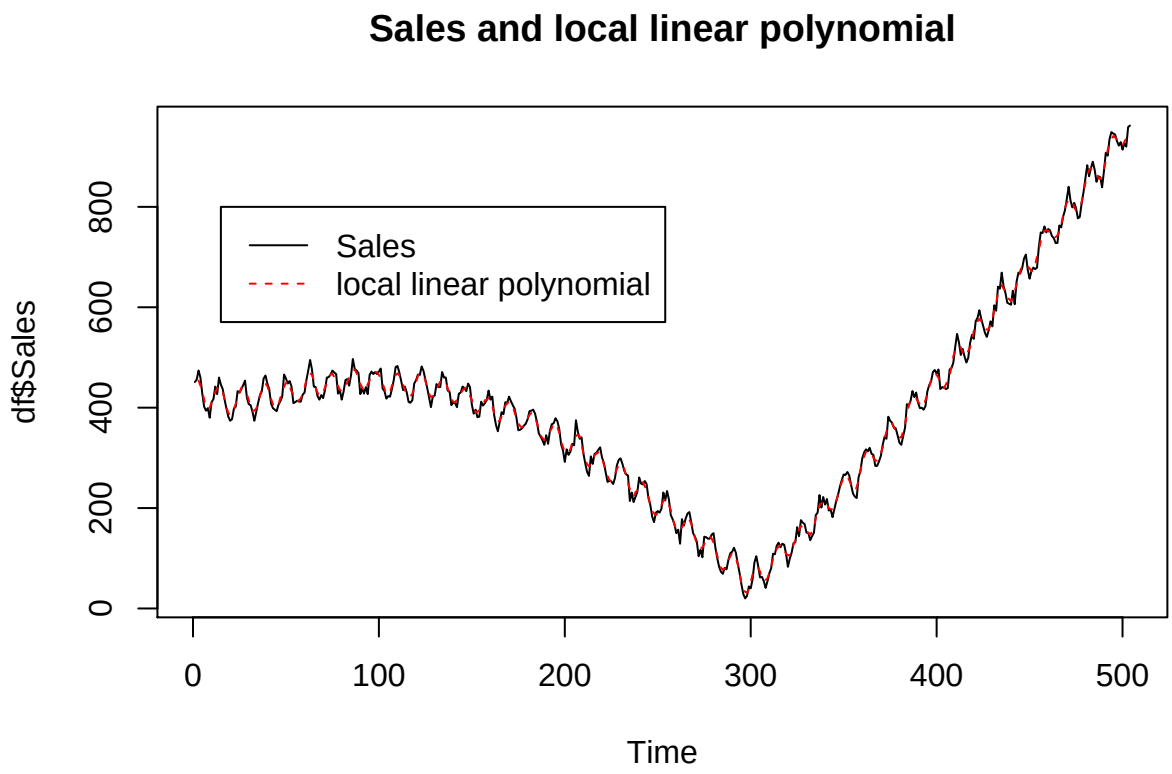
Thus we find $\hat{m}_t = \hat{\beta}_0 = \frac{1}{5}(x_{t-2} + x_{t-1} + x_t + x_{t+1} + x_{t+2})$. We can also calculate $\hat{\beta}_1$ but since we know $\hat{m}_t = \hat{\beta}_0 + \hat{\beta}_1(0) = \hat{\beta}_0$ the trend functon $\hat{m}_t \ \forall \ t$ we have no need.

QUESTION 2 b)

We now compute the value of $\hat{m}_t$ at all points and plot it against the actual values

```
##Local linear model fitting
##Initialise vector
m <- (1:length(df$Sales))
##remove first two and last two data points since we cannot fit a local trend around them
##(For 5 point local linear model)
for (i in c(1,2,length(df$Sales)-1,length(df$Sales))){
  m[i] = NA
}

##Find the trendfor all T
##t = 3 is the first valid data point so we start there and iterate untill the 3rd to last
t <- 3

while(t<length(df$Sales)-1){
  m[t] <- (df$Sales[t-2]+df$Sales[t-1]+df$Sales[t]+df$Sales[t+1]+df$Sales[t+2])/5
  t <- t+1
}
##Add the trend to the dataframe
df[,3] <- m
colnames(df) <- c("Sales","QuadFitValues","LocalPolyValues")

##Lets plot the time series and local linear fit. pch = ... specifies the
##type of dot to be used in the plot
ts.plot(df$Sales)

lines(1:length(df$Sales),
      df$LocalPolyValues,
      lty=2,
      col = "red")

##adding a title and legend for clarity
title("Sales and local linear polynomial")
legend(15,800, legend = c("Sales","local linear polynomial"),
```

```
       lty =c(1,2),
       col = c("black","red"))
```



QUESTION 3 (Analysis)

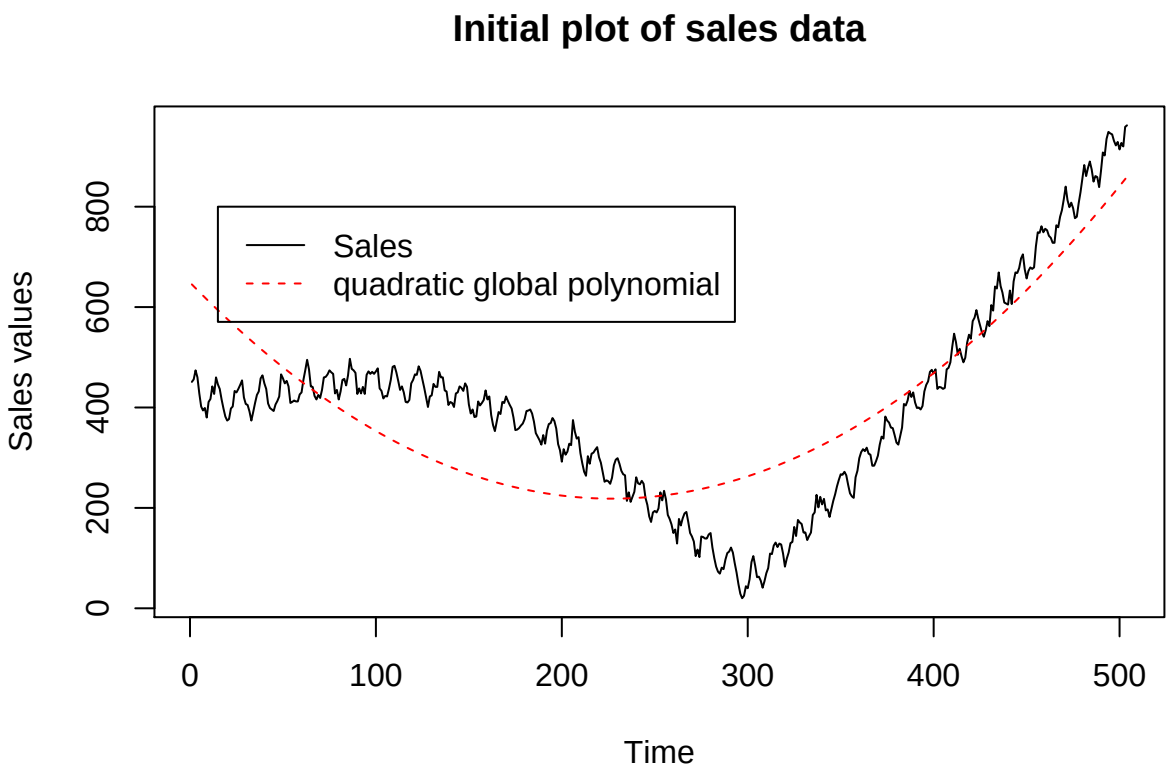
We would like to choose the model which best suits the data, so we now consider the residuals of each model. From the residuales R_i we expect the following properties

$$\sum_i R_i = 0, \mathbb{E}[R_i] = 0$$

among others we are currently not interested in.

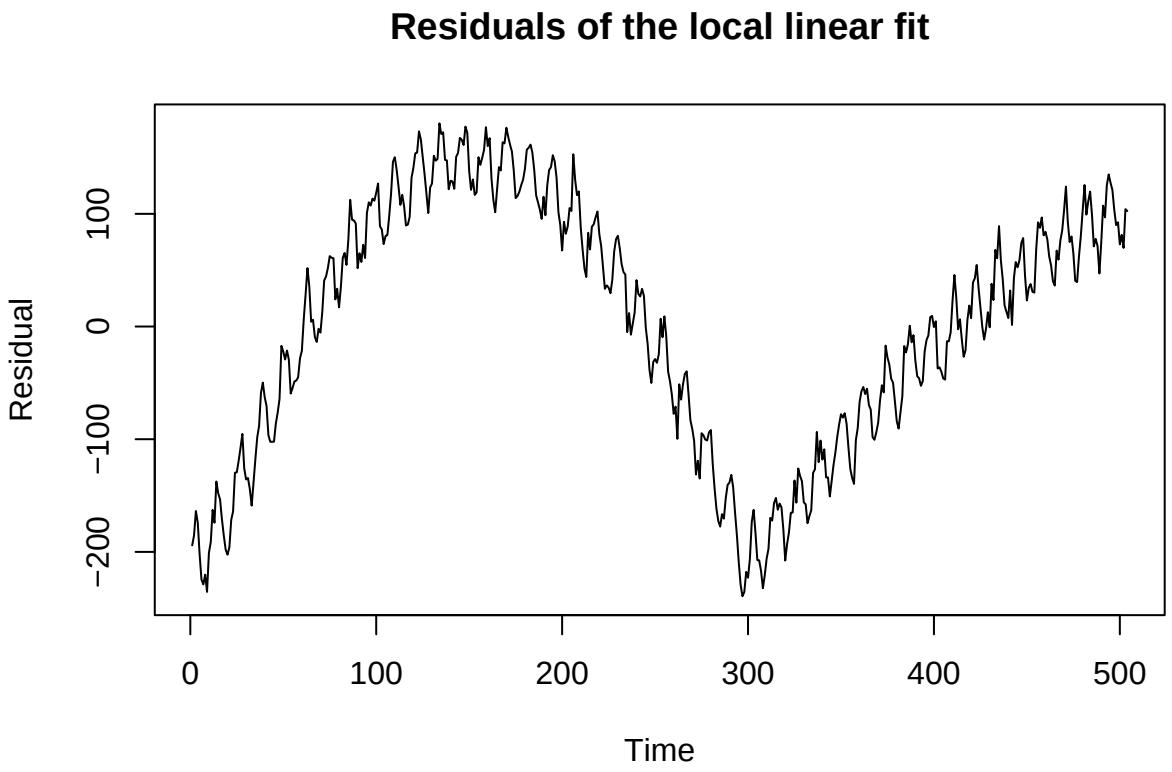
We hope to see only random motion and seasonality remaining.

```
##Plot the data
ts.plot(df$Sales,
        xlab = 'Time',
        ylab = 'Sales values')
title("Initial plot of sales data")
lines(quadfit$fitted.values,
      col = "red",
      lty = 2)
legend(15,800,
       legend = c("Sales","quadratic global polynomial"),
       lty =c(1,2),
       col = c("black","red"))
```



The first point I'll make is that we are in no danger of overfitting. The global polynomial fits the graph for very few values of t, and for the few that it does do a reasonable job of predicting, its is but a fleeting coincidence. Its fair to say that if there is a global polynomial then it is not a quadratic one. Its possible we would have more luck modeling the data with a quartic or cubic (or higher order) polynomial, but as it stands this method significantly underfits the given data and will be very unlikely to be able to predict future data points.

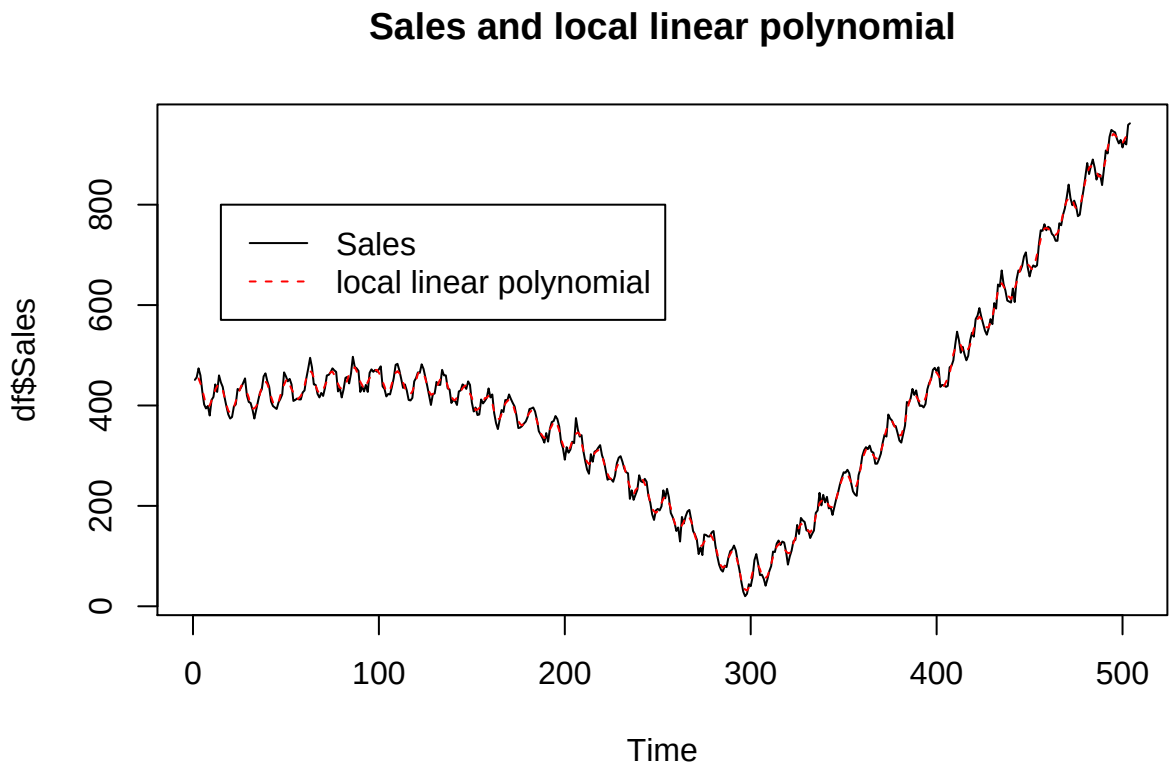
```
##residue of the quadratic fit
df$QuadFitRes = df$Sales - df$QuadFitValues
ts.plot(df$QuadFitRes,
        xlab = "Time",
        ylab = "Residual",
        main = "Residuals of the local linear fit")
```



We can clearly see that what is left in the residual is not simply a seasonal and random component, the trend still has considerable effect on the residue of the global polynomial fit, therefore this fit doesn't represent a good model for the given data.

We now look at the Local Linear polynomial:

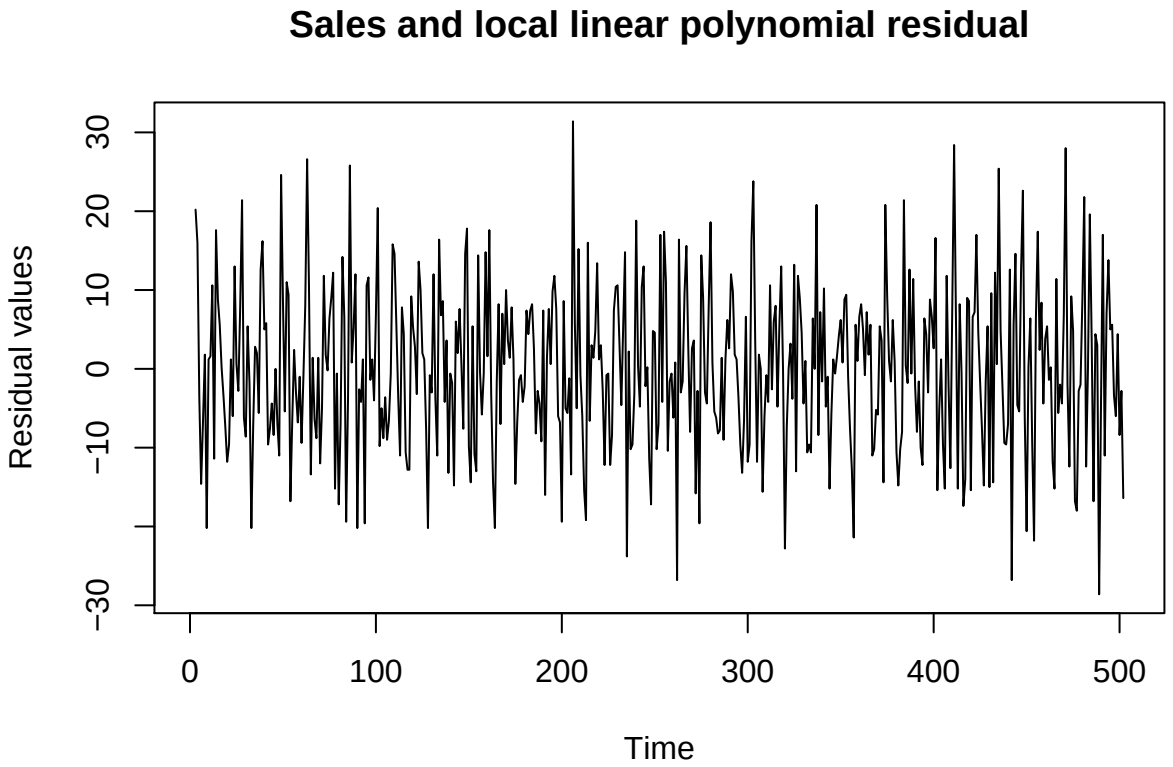
```
##Lets plot the time series and local linear fit.
##pch = ... specifies the type of dot to be used in the plot
ts.plot(df$Sales)

lines(1:length(df$Sales),df$LocalPolyValues,lty=2,col = "red")

##adding a title and legend for the sake of it!
title("Sales and local linear polynomial")
legend(15,800,
       legend = c("Sales","local linear polynomial"),
       lty =c(1,2),
       col = c("black","red"))
```



This looks much better, the local linear polynomial does a much better job of following the data. However, it has also captured the seasonality of the time series, which suggests that the model could have overfit the data, that is to say the model may have absorbed the random movement of the data and will thus not be a good fit for future data. Its hard to say to what degree this has happened but a better fit would trace the movement of the data without incorporating the seasonal variations.

Consider the residue of the local linear polynomial fit given here

```
##We remove the trend component of the sales data.
df$PolyFitRes = df$Sales - df$LocalPolyValues
ts.plot(df$PolyFitRes,
        xlab = "Time",
        ylab = "Residual values")
title("Sales and local linear polynomial residual")
```



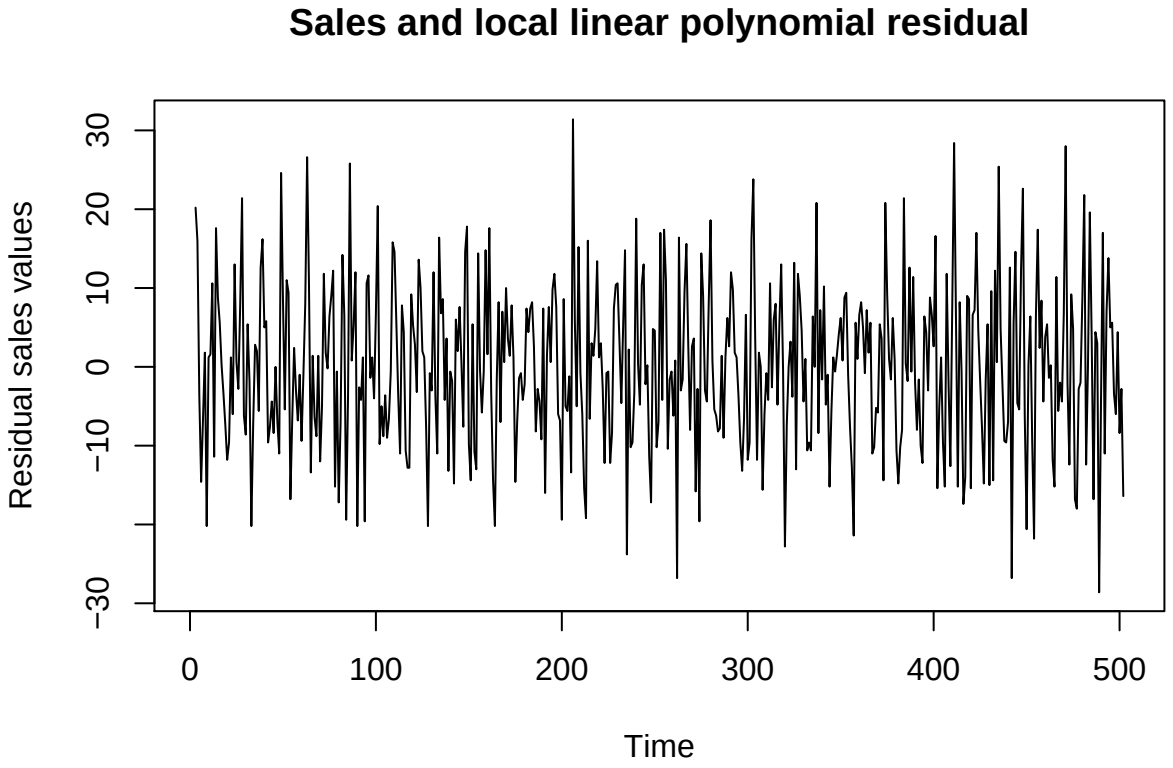
The remaining data is roughly centered around 0 with what appears to be periodic fluctuations in the data. Its believable that this could be the seasonal and white noise components, which implies that the local linear polynomial fit matches the trend well.

QUESTION 4 a)

We now endeavor to determine the seasonal behavior of the data, assuming that the trend is given by the local linear polynomial as discussed above.

We begin by computing the residue $\hat{y}_t = x_t - \hat{m}_t = s_t + Z_t$ to separate the trend from the seasonal and random components of the data.

```
##We have removed the trend component of the sales data in the last question.
##We plot the polynomial fit residual
ts.plot(df$PolyFitRes,
        ylab = "Residual sales values",
        xlab = "Time")
title("Sales and local linear polynomial residual")
```



We are given that the seasonality has period of 12, so we know that for the time series $Y_t = \hat{s}_+ Z_t$ we have $Y_{i+ju} = s_i + Z_{i+ju}$ where $i < u,\ u = 12$ the time period, $i, j \in \mathbb{Z}$ To ensure that our model of s_t is the best possible we use the least squares cost function, $S = \sum_{i-1}^{n}(Y_i - s_i)^2 = \sum_{j=0}^{\frac{n}{12}-1}\sum_{i=1}^{12}(Y_{i+uj} - s_i)^2$ by taking the derviative w.r.t s_i we dobtain

$$\frac{dS}{ds_i} = -2\sum_{j=0}^{\frac{n}{12}-1}(x_{12j+i} - s_i) = 0 \implies \sum_{j=0}^{\frac{n}{12}-1}(x_{12j+i} - s_i) = 0$$

$$\sum_{j=0}^{\frac{n}{12}-1} x_{12j+i} = \sum_{j=0}^{\frac{n}{12}-1} s_i = \frac{12}{n}s_i$$

therefore

$$s_i = \frac{12}{n}\sum_{j=0}^{n/12-1} x_{12j+i}$$

which is the average of the ith, i+jth, i+2jth ,... etc components. Hence the best seasonal components are given by

$$s_i = mean(x_i, x_{i+12}, ..., x_{i+\frac{n}{12}j})$$

for $i = 1, ..., 12$.

QUESTION 4 b)

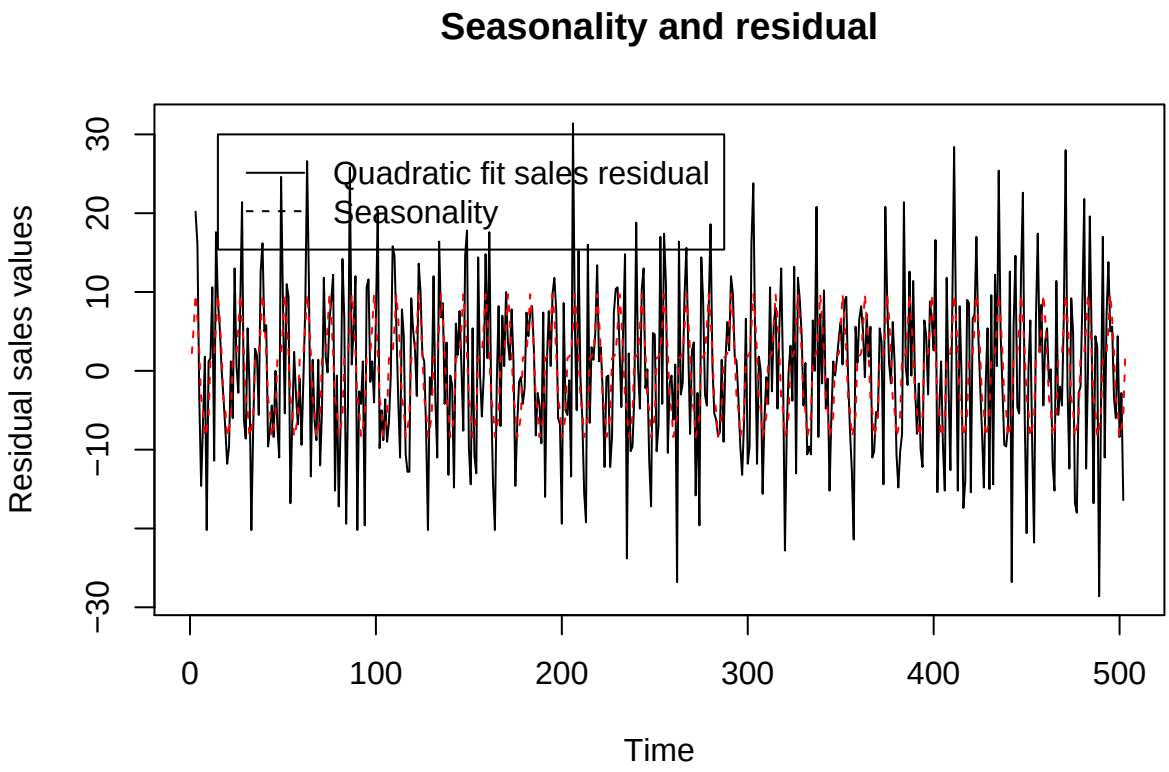
We now compute the value of the 12 seasonal components

```
##Since m_t have NAs for the first and last two entries we must deal with these data
##points in some way. I will choose to exclude
##them completely since I feel it makes no sense to make up data when excluding them
##changes very little. I will use na.rm = TRUE when
##computing the mean.
```

```
##Create a repeating index of length 12
i <- rep_len(1:12, length(df$PolyFitRes))
##Find the mean for each of the 12 months
p <- c(mean(df$PolyFitRes[i==1],na.rm = TRUE),
       mean(df$PolyFitRes[i==2],na.rm = TRUE),
       mean(df$PolyFitRes[i==3],na.rm = TRUE),
       mean(df$PolyFitRes[i==4],na.rm = TRUE),
       mean(df$PolyFitRes[i==5],na.rm = TRUE),
       mean(df$PolyFitRes[i==6],na.rm = TRUE),
       mean(df$PolyFitRes[i==7],na.rm = TRUE),
       mean(df$PolyFitRes[i==8],na.rm = TRUE),
       mean(df$PolyFitRes[i==9],na.rm = TRUE),
       mean(df$PolyFitRes[i==10],na.rm = TRUE),
       mean(df$PolyFitRes[i==11],na.rm = TRUE),
       mean(df$PolyFitRes[i==12],na.rm = TRUE))
df$PolyFitSeasonality <- rep_len(p, length(df$PolyFitRes))
##Lets view the seasonal mean values for each time value
dummy = 1:12
```

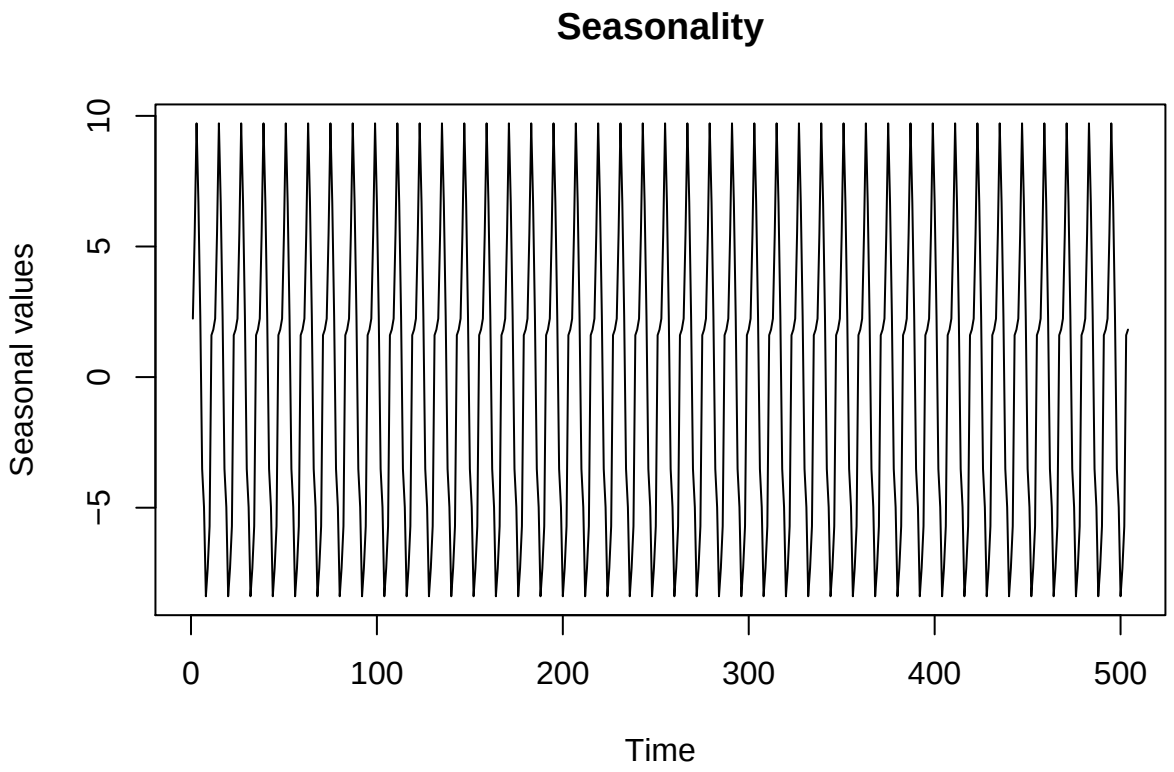
Lets plot the seasonality with the residue and with the trend to see if it has improved our model

```
ts.plot(df$PolyFitRes,
        ylab = "Residual sales values",
        xlab = "Time")
lines(1:length(sales),
      df$PolyFitSeasonality,
      lty=2,
      col = "red")
legend(15,30,
       legend = c("Quadratic fit sales residual","Seasonality"),
       lty =c(1,2))
title("Seasonality and residual")
```



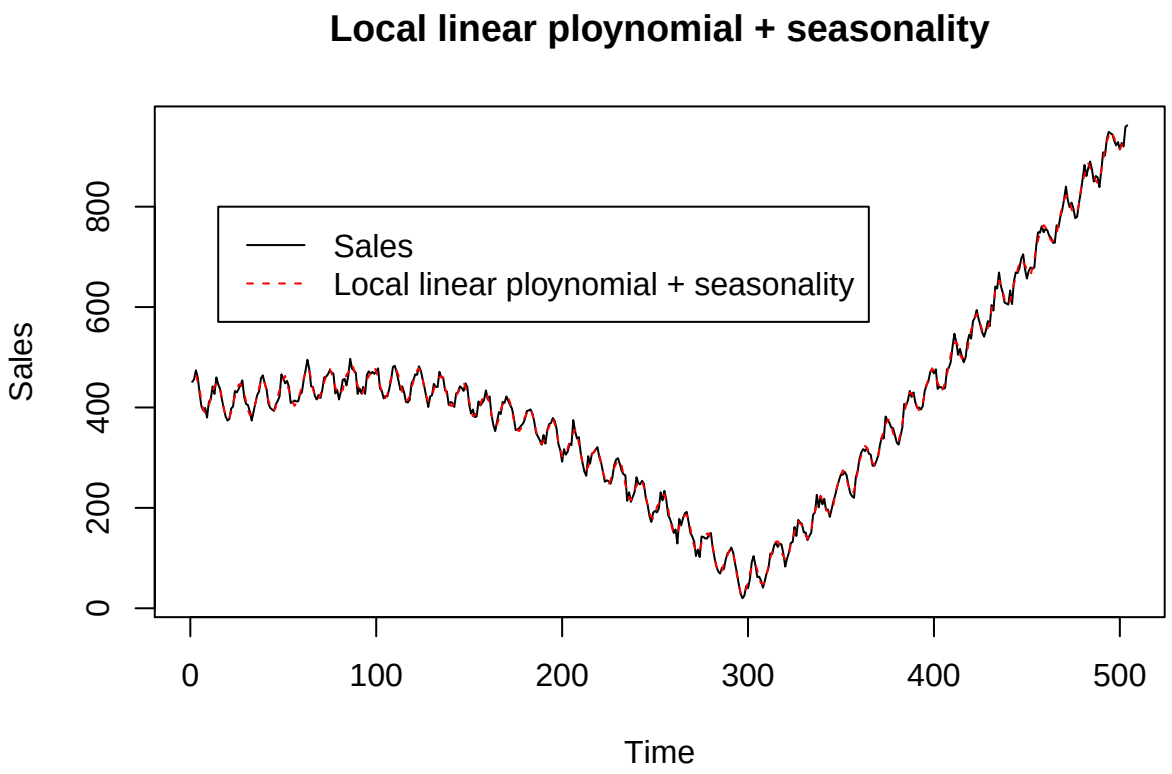
Since it is hard to distinguish between the residual and the seasonality, here is just the seasonality.

```
ts.plot(df$PolyFitSeasonality,
        ylab = "Seasonal values",
        xlab = "Time")
legend(15,30,
       legend = c("Seasonality"),
       lty =c(1,2))
title("Seasonality")
```



We now plot the full model, including both the trend and seasinality.

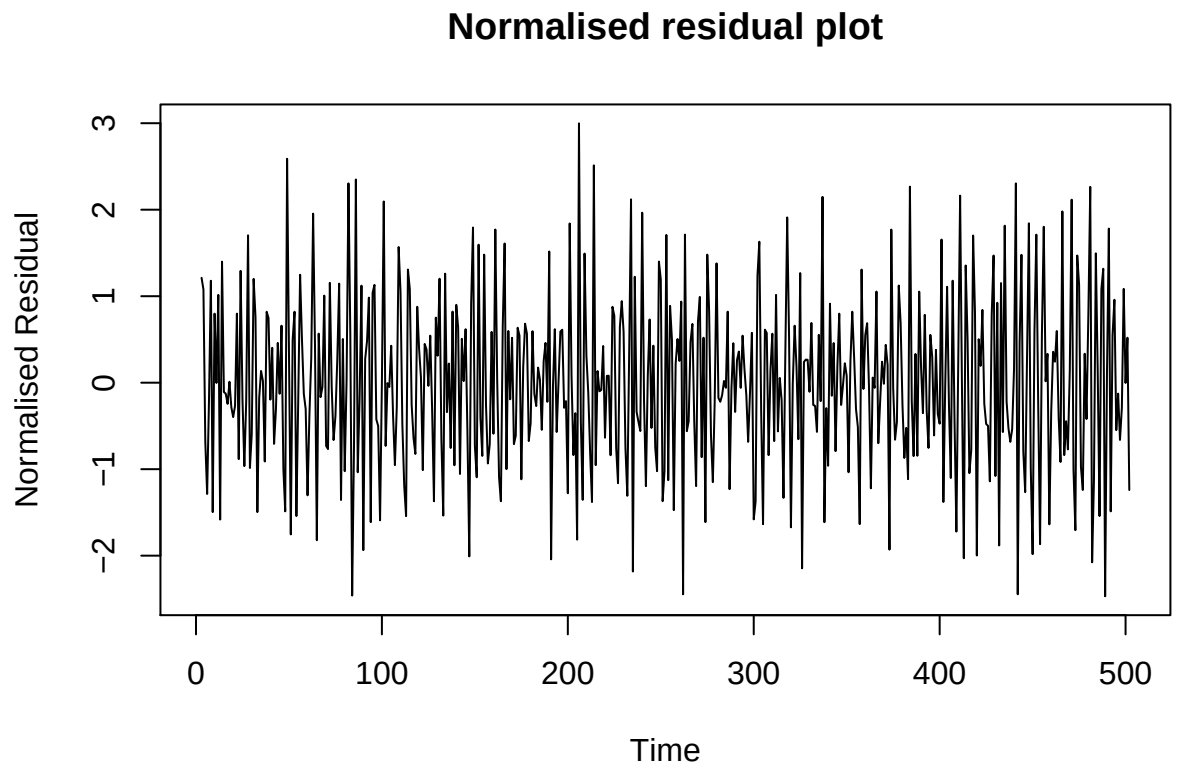
```
ts.plot(sales,
        xlab = "Time",
        ylab = "Sales")
lines(1:length(sales),
      df$LocalPolyValues+df$PolyFitSeasonality,
      lty=2,
      col = "red")
legend(15,800,
       legend = c("Sales","Local linear ploynomial + seasonality"),
       lty =c(1,2),
       col = c("black","red"))
title("Local linear ploynomial + seasonality")
```



QUESTION 4 c)

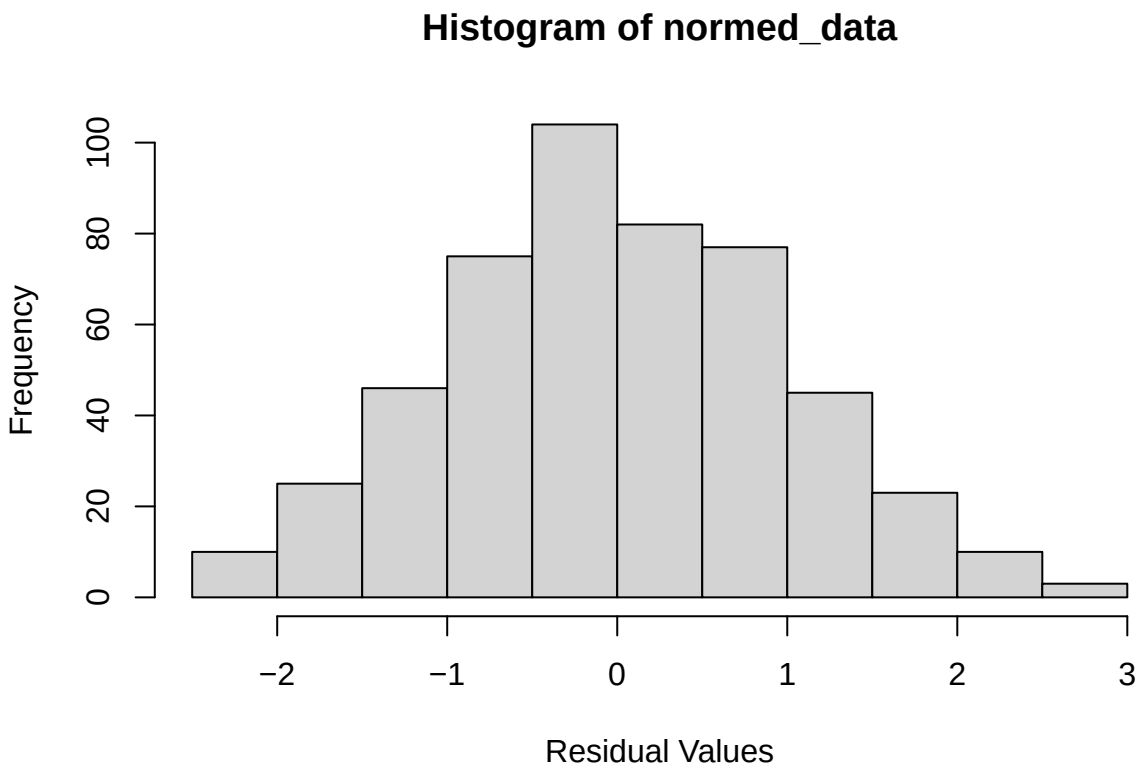
To check if what we have found is a good model for the sales values then we wish to know the residual given by $x_t - m_t - s_t$. If this remaining signal is a white noise process then we can say that this is a good fit and evaluate if the model is over or under-fitting the data.

```
##Compute the white noise
df$WhiteNoiseLocalPolySeasonality = df$Sales - df$LocalPolyValues - df$PolyFitSeasonality
##Take the mean and variance of the data, we then normalise the data
##and visualise the distribution
E = mean(df$WhiteNoiseLocalPolySeasonality[3:length(df$WhiteNoiseLocalPolySeasonality)-3],
         na.rm = TRUE)
var = var(df$WhiteNoiseLocalPolySeasonality[3:length(df$WhiteNoiseLocalPolySeasonality)-3],
          na.rm = TRUE)
normed_data = (df$WhiteNoiseLocalPolySeasonality - E)/sqrt(var)

##We check the s.d of the data to see if it is likely from the normal distribution.
ts.plot(normed_data, ylab = "Normalised Residual", xlab = "Time")
title("Normalised residual plot")
```



It is hard to see any patterns from this plot, so considering the histogram of the data to see if it has an underlying distribution may be more useful.

```
hist(normed_data, xlab = "Residual Values")
```



From the residue plot and histogram, we see that much of our data is roughly centred around 0 and within a

few standard deviations of the mean $\mu = 0$ after being normalised, this implies that most of the remaining data is white noise and (possibly) stems from the normal distribution.

To confirm that these are the true residuals and that our model is a good fit for the data we compute the sum and mean of the data, we expect that both are approximately 0.

```
##Mean of residues
mean(normed_data,na.rm = TRUE)
```

```
## [1] -0.002482523
```

```
##Sum of residuals
sum(normed_data, na.rm = TRUE)
```

```
## [1] -1.241261
```

So the mean of the residuals is $\mathbb{E}[R] = -0.00248$ and the sum is -1.24. I believe these are close enough to 0 to be the true values. Hence the model encodes a suitable amount of information to be a good approximation of the true trend and seasonality. We only need to be careful of over fitting the data.

It would be good to improve the model, however if we increase the degree of the polynomial we are using we risk over fitting the data. The model we currently have does a good job fitting the data, with a low risk of over fitting.

QUESTION 5)

Since a local or global polynomial can only fit trends that are locally or globally polynomial, it would be good to try a different type of model all together. An exponentially weighted moving average (EWA) is less restrictive than a polynomial, but introduces a lag, so may perform poorly when a trend changes rapidly as in our example. This can be improved by using the Holt-Winter method. The EWA is of the form

$$\hat{m}_t = \alpha_t x_t + (1-\alpha)\hat{m}_{t-1}$$

The parameters we need to choose are α, and $\hat{m}_0$. α is the weight of the current time series value and is between 1 and 0. $\hat{m}_0$ is not particularly important because it is quickly dominated by the other terms.

We can see this is we write

$$\hat{m}_t = \alpha_t x_t + (1-\alpha)\hat{m}_{t-1} = \sum_{n-0}^{t-1}[\alpha(1-\alpha)^n x_{t-n}] + (1-\alpha)^t \hat{m}_0$$

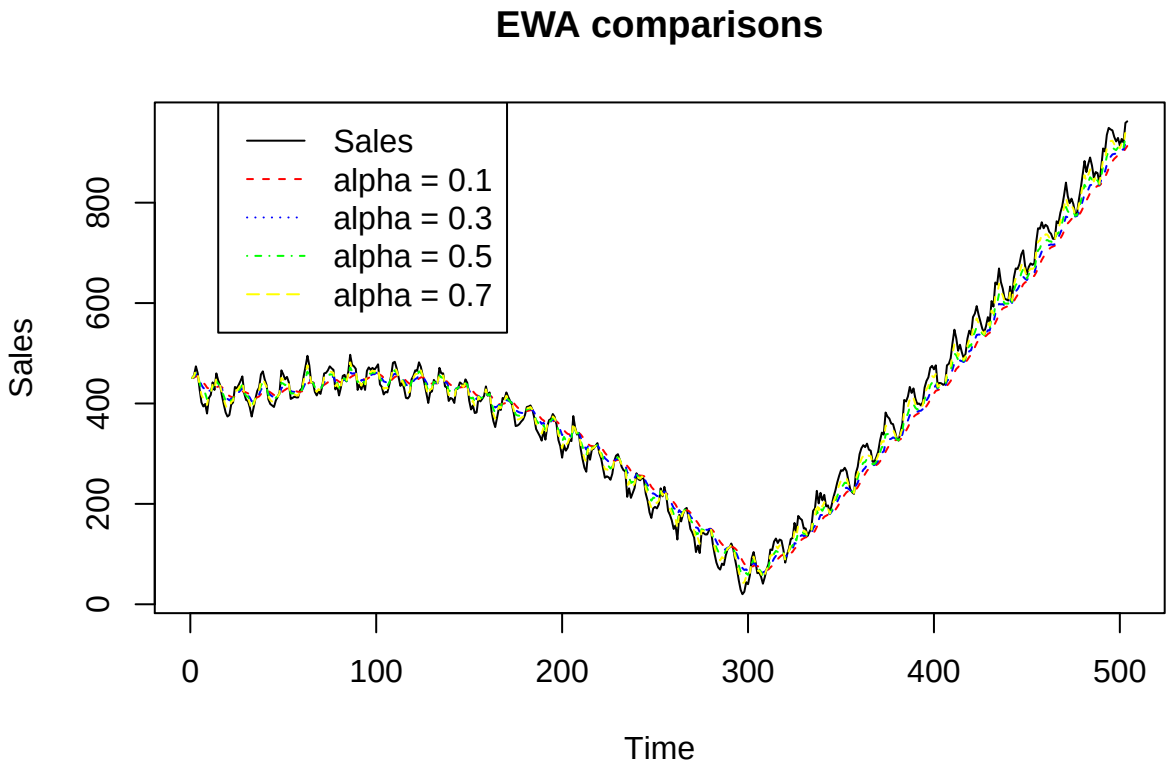
Since α is smaller that 1 the last term goes to 0 very quickly and is irrelevant after only a few iterations.

We will create 9 EWAs and compare the quality of the fit. We begin by generating 9 α values between 0 and 1, then we generate the sequences and plot then. We will then consider the residuals of the most promising values of α.

```
alpha_E= seq(0.1,0.9,1/10) #Define the values of alpha
Model <- data.frame(matrix(NA,
                           nrow = 9,
                           ncol = length(df$Sales))
                    ) #Empty data frame to store our 9 models.
M0 = df$Sales[1]      #We set M0 to be the first value of Sales.
                      #This is to prevent a large tail from forming at the start,
for (A in (1:length(alpha_E))){ #Loop over alpha
  Model[A,1] = M0     #Set initial condition
  for (i in c(2:length(df$Sales))){  #Loop over df$Sales
        Model[A,i] = alpha_E[A]*df$Sales[i] + (1-alpha_E[A])*Model[i-1]
    }
}
```

Lets plot the 1st, 3rd, 5th and 7th models. We expect the models with higher α values to adapt more quickly and thus be more useful. Which is exactly what we see below.

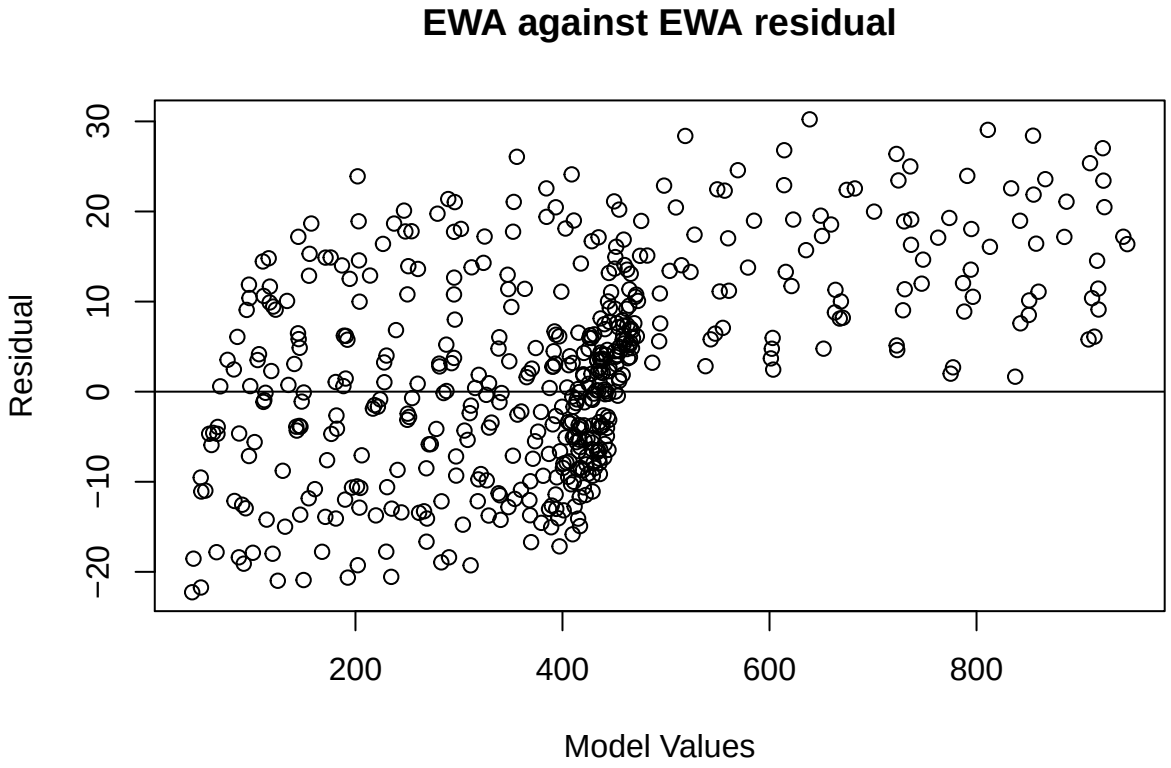
```
ts.plot(df$Sales,ylab = "Sales")
Dummy = Model          #Create a Dummy for the Model dataframe
lines(1:length(df$Sales),
      Dummy[1,],
      lty=2,
      col = "red")
lines(1:length(df$Sales),
      Dummy[3,],
      lty=2,
      col = "blue")
lines(1:length(df$Sales),
      Dummy[5,],
      lty=2,
      col = "green")
lines(1:length(df$Sales),
      Dummy[7,],
      lty=2,
      col = "yellow")

##adding a title and legend for clarity
title("EWA comparisons")
legend(15,1000, legend = c("Sales",
                           "alpha = 0.1",
                           "alpha = 0.3",
                           "alpha = 0.5",
                           "alpha = 0.7"),
       lty =c(1,2,3,4,5),
       col = c("black",
               "red",
               "blue",
               "green",
               "yellow"))
```



The fit looks good on all these models, but past time 300, the EWA consistently under guesses the real values. To check this we can compare the residues to the fitted values.

[I appreciate that the plot might be difficult to see, but the residual below is what tells us the story]

```
t = Model[7,] - df$Sales
plot(c(Model[7,]),c(df$Sales - Model[7,]), xlab = "Model Values", ylab = "Residual")
title("EWA against EWA residual")
abline(h=0)
```

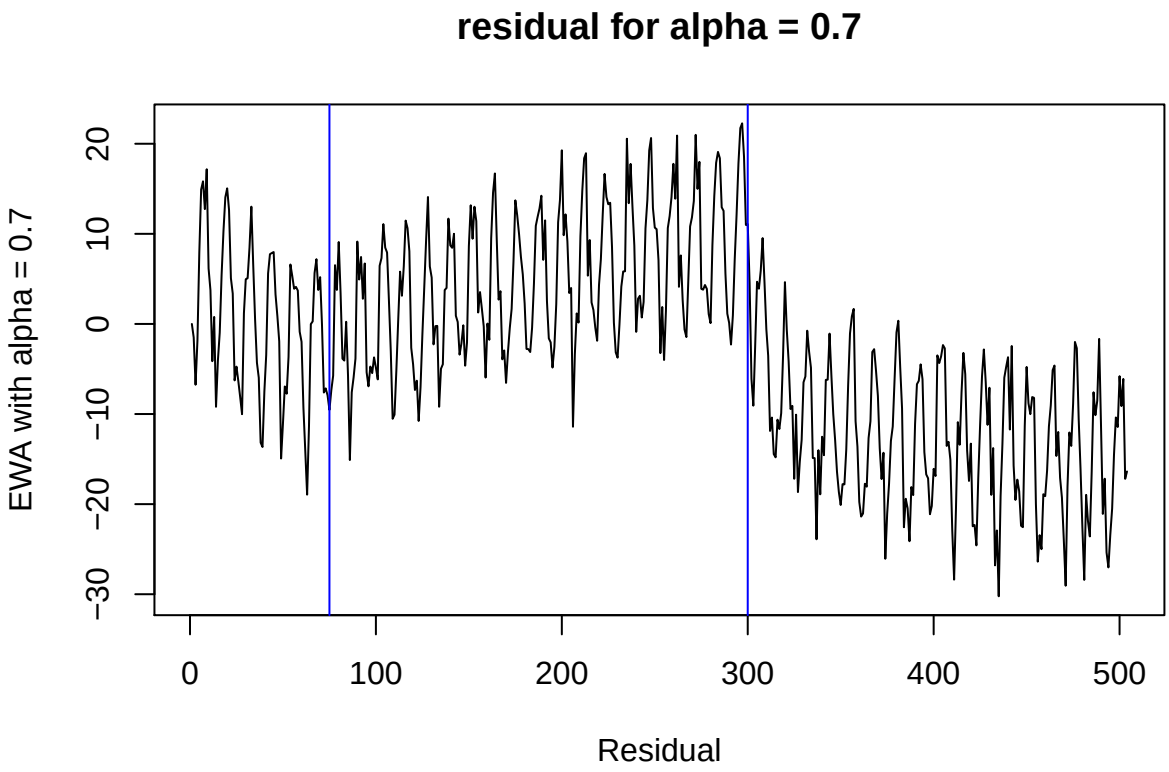


For low fitted values (fitted values between 0 and 400) the model fits the data well, there is no obvious correlation. For fitted values around 500 and higher however, the model significantly under guesses the real data. As can be seen by the residual values all being above 0 for model values above 500.

As expected we can see that the larger α is the better the fit is. This makes sense because the current time series values has a higher impact on the score at any time.

To see how good the fit is we consider the residual of each EWA. Due to the bulkiness of the plots, I will only show it for $\alpha = 0.7$. For the rest check the appendix.

```
row = c(Model[7,] - df$Sales)
ts.plot(row,
        xlab = "Residual",
        ylab = "EWA with alpha = 0.7" )
title("residual for alpha = 0.7")
abline(v=75, col="blue")
abline(v=300, col="blue")
```



For each model we see distinct changes in the residuals at the locations marked by blue lines, at Time equal to 75 and 300. This might be an indicator that the model is not sensitive enough to sudden change. To improve this we might increase α further but we would then risk over-fitting the data, rendering the model little more than a statement of the obvious.

A second approach is the Holt-Winters method. We use an extra term to determine how fast the trend is changing.

$$\hat{\omega} = \alpha\hat{x}_t + (1-\alpha)(\omega_{t-1} + T_{t-1})$$
$$T_t = \beta(\omega_t - \omega_{t-1})$$

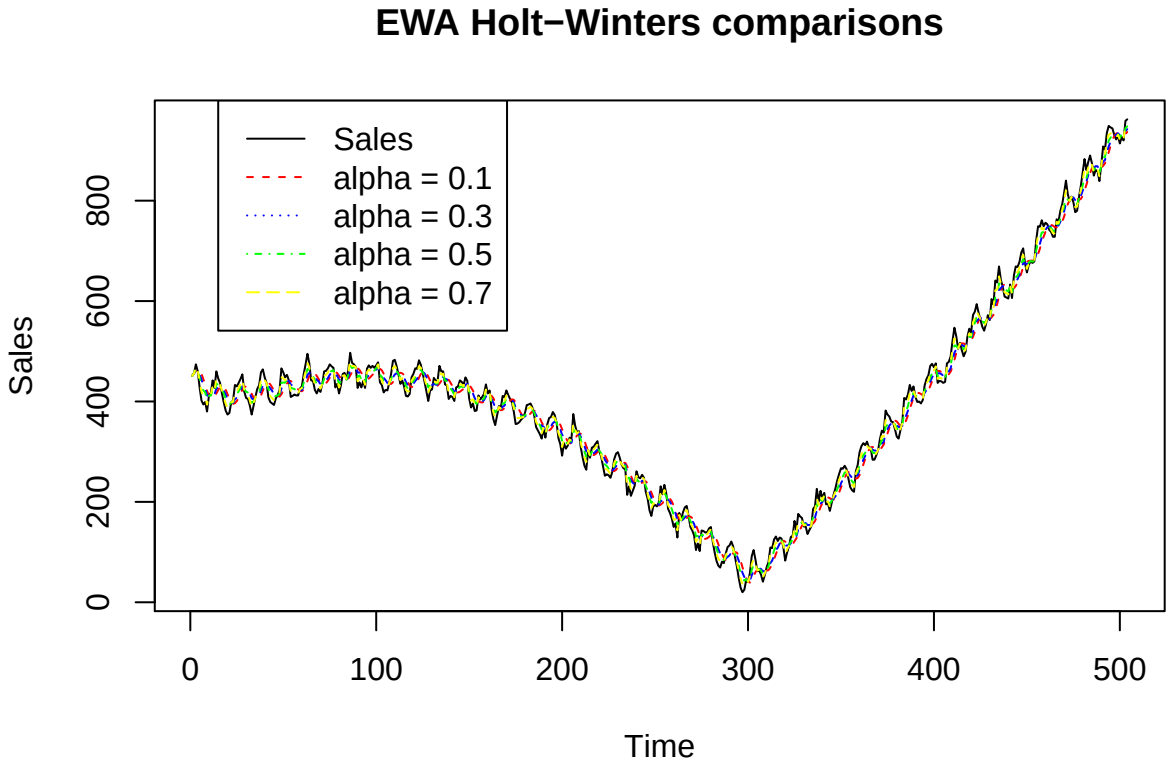
Where T is the change in the trend at that point. To set $\beta = 0.7$ which should give us some improvement.

As weith EWA we generate the models for 9 values of α.

```
alpha_H= seq(0.1,0.9,1/10) #Define the values of alpha
Holt_Winters <- data.frame(matrix(NA,
                                  nrow = 9,
                                  ncol = length(df$Sales))) #Empty data frame to store our 9 models.
W0 = df$Sales[1] #We set M0 to be the first value of Sales.
                 #This is to prevent a large tail from forming at the start,
W1 = df$Sales[2] #We need a second data point for this model, so we use the second value of sales.
BETA = 0.7
for (A in (1:length(alpha_H))){ #Loop over alpha
  Holt_Winters[A,1] = W0 #Set initial condition
  Holt_Winters[A,2] = W1
  for (i in c(3:length(df$Sales))){  #Loop over df$Sales
        Holt_Winters[A,i] = alpha_H[A]*df$Sales[i] + (1-alpha_H[A])*(Holt_Winters[i-1]+BETA*(Holt_Winte
    }
}
```

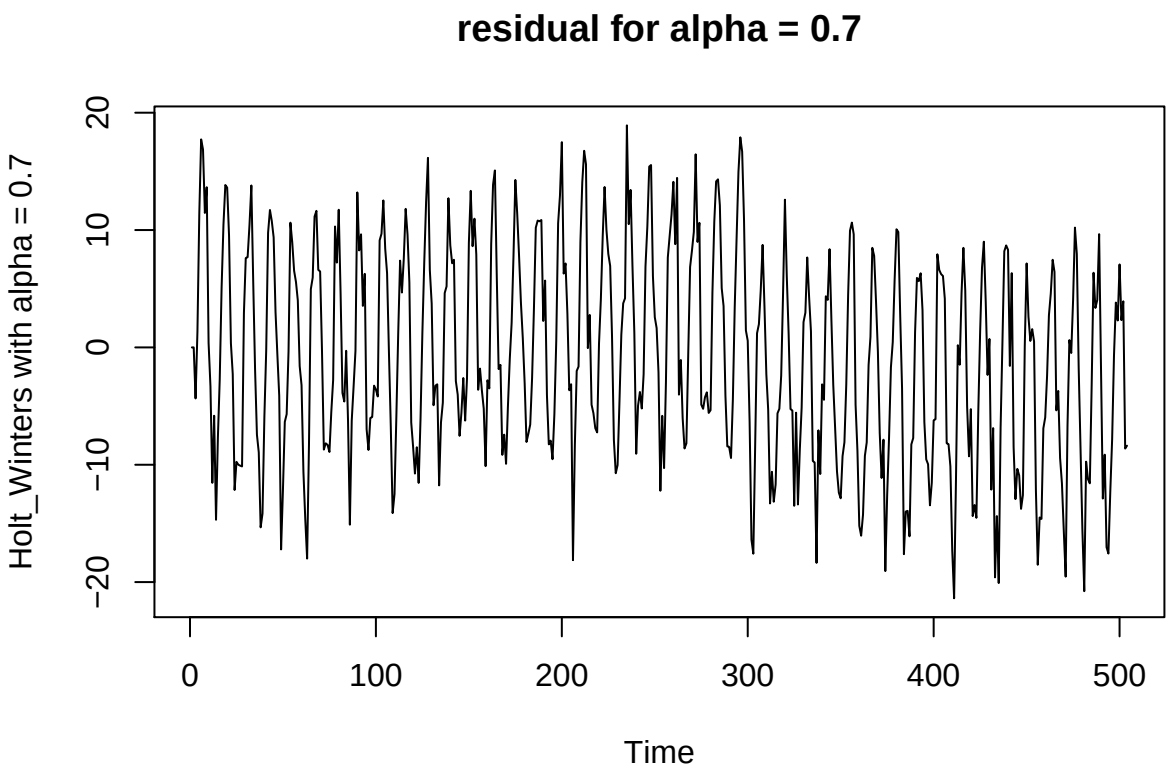
Again we plot 4 of the Holt-Winters models against the sales.

```
##Plotting Holt-Winters
ts.plot(df$Sales,ylab = "Sales")
lines(1:length(df$Sales),
      Holt_Winters[1,],
      lty=2,
      col = "red")
lines(1:length(df$Sales),
      Holt_Winters[3,],
      lty=2,
      col = "blue")
lines(1:length(df$Sales),
      Holt_Winters[5,],
      lty=2,
      col = "green")
lines(1:length(df$Sales),
      Holt_Winters[7,],
      lty=2,
      col = "yellow")

##adding a title and legend for clarity
title("EWA Holt-Winters comparisons")
legend(15,1000, legend = c("Sales",
                           "alpha = 0.1",
                           "alpha = 0.3",
                           "alpha = 0.5",
                           "alpha = 0.7"),
       lty =c(1,2,3,4,5),
       col = c("black",
               "red",
               "blue",
               "green",
               "yellow"))
```



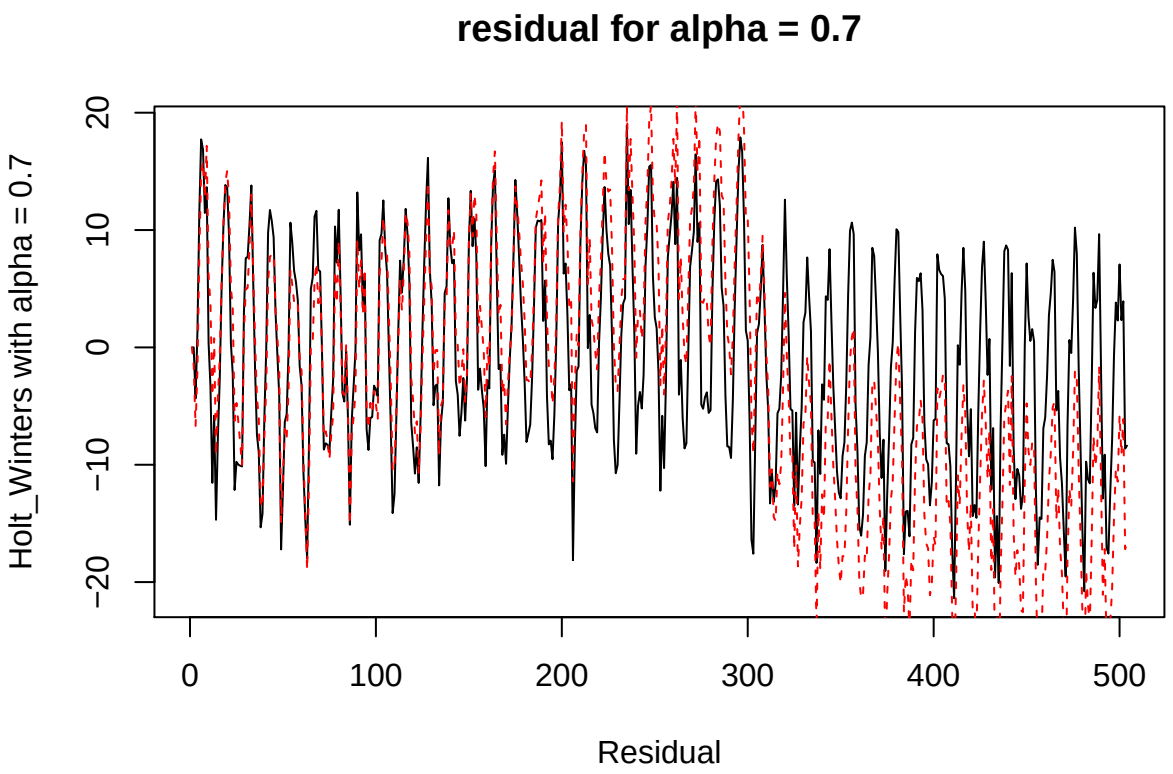
Already these look better than just the EWAs. To see exactly the difference we can compare their residuals. We will only consider the case for $\alpha = 0.7$

```
row = c(Holt_Winters[7,] - df$Sales)
ts.plot(row,
        xlab = "Time",
        ylab = "Holt_Winters with alpha = 0.7" )
title("residual for alpha = 0.7")
```



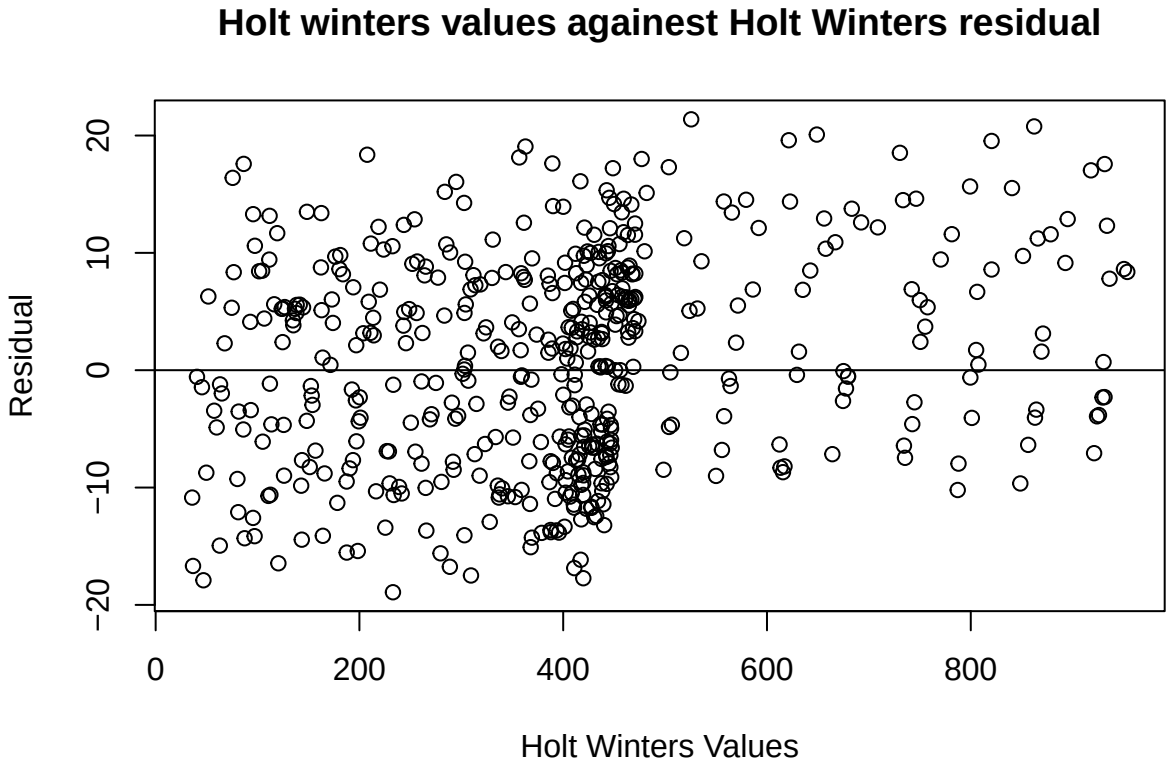
Even to the eye this looks more centred around 0 which is exactly what we want to see happening. We now overlay the EWA residual to verify this.

```
row = c(Holt_Winters[7,] - df$Sales)
ts.plot(row,
        xlab = "Residual",
        ylab = "Holt_Winters with alpha = 0.7" )
title("residual for alpha = 0.7")
lines(1:length(df$Sales),
      Model[7,]-df$Sales,
      lty=2,
      col = "red")
```



Both EWA and Holt-Winters perform similarly up to around Time = 300, after this the Holt-Winters methods out performs the EWA significantly. Even with the irregular residuals, the EWA and Holt-Winters models both out perform the local and global quadratic polynomials.

```
t = Holt_Winters[7,] - df$Sales
plot(c(Holt_Winters[7,]),c(df$Sales - Holt_Winters[7,]), xlab = "Holt Winters Values", ylab = "Residual
title("Holt winters values againest Holt Winters residual")
abline(h=0)
```



This is also much better, we see that the distribution of the residual has improved, with far fewer under-guessing event. This suggests that the Holt winters model is an improvement over just EWA.

To conclude, EWA and Holt-Winters both give residuals with lower magnitude than the local linear polynomial fit, but EWA and Holt-Winter both have elements of the trend remaining within them. This is because EWA and Holt-Winter are both slower to adapt to sudden changes, which are common in this particular dataset. Despite this Holt Winters is a significant improvement over EWA, particularly for fitted vales greater than 500.